SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

RINGWIZARD

Websovellus lisätyövuorojen hallintaan

TEKIJÄ: Joona Ollikainen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma/Tutkinto-ohjelma Tietotekniikan tutkinto-ohjelma	
Työn tekijä(t) Joona Ollikainen	
Työn nimi Ringwizard – Websovellus Lisätyövuorojen Hallintaan	
Päiväys 26.5.2020	Sivumäärä/Liitteet 27
Toimeksiantaja/Yhteistyökumppani(t) Jussi Nivamo DigiCenter	

Tiivistelmä

Tämän opinnäytetyön tarkoituksena oli kehittää websovellus lisätyövuorojen hallintaan, jossa työntekijä voi hallinnoida ja ottaa lisävuoroja joko yrityksen sisäisenä tai ulkoisena työntekijänä. Opinnäytetyötä kehitettiin omalle toiminimelle, mikä perustettiin syksyllä 2019. Sovellus toimii samalla korvaavan työntekijän etsimisessä esimerkiksi äkillisen sairastapauksen kohdalla, jolloin turhat puhelinsoitot korvataan nopealla työntekijähaulla ja nopealla viestinnällä sopivan henkilön löydyttyä työtehtävään.

Opinnäytetyö aloitettiin suunnittelemalla käyttöliittymä sekä tarvittavat toiminnot, jotta sovellusta pystyttiin testaamaan. Opinnäytetyössä hyödynnettiin .NET Framework MVC mallia. Ohjelmointikielenä toimi C#. Muut toiminnot toteutettiin SignalR tekniikalla ja Nexmo SMS API rajapintaa hyödyntäen.

Tämän opinnäytetyön tuloksena valmistui toimiva POC – malli, jota pystyy käyttämään ja testaamaan. Opinnäytetyön keskeisimmät toiminnot ovat hakea työntekijää rekisteröityjen käyttäjien joukosta esimerkiksi lääkelupien perusteella. Sovellus mahdollistaa ilmoitusten lähettämisen vapaista vuoroista työntekijöille. Hyväksytty vuoro tallentuu käyttäjälle sekä hallinnointi puolelle henkilökohtaiseen kalenteriin. Hallinnointi puolen on mahdollista tehdä lisätyövuoro pyyntöjä, jotka siirtyvät kalenteriin tyhjinä vuoroina. Käyttäjän hyväksyttyä vuoron se siirtyy varatuksi vuoroksi käyttäjälle.

Avainsanat
Azure, Nexmo, MVC, Web applikaatio

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Joona Ollikainen			
Title of Thesis Ringwizard – Web Application To Control Extra Shifts			
Date	26 May 2020	Pages/Appendices	27
Client Organisation /Partners Jussi Nivamo DigiCenter			

Abstract

The purpose of this thesis was to develop a web application for managing additional shifts, where an employee can manage and take additional shifts either as an internal or external employee of the company. The thesis was developed under the author's own business name, which was established in the autumn of 2019. The application also works in finding a replacement employee, for example in the event of a sudden illness, replacing unnecessary phone calls with quick employee search and quick communication.

The thesis started with designing the user interface and the necessary functions to be able to test the application. The .NET Framework MVC model was utilized in the thesis. The programming language was C #. Other functions were implemented using SignalR technology and using the Nexmo SMS API.

As a result of this thesis, a functional POC model was completed, with which the software could be used and tested. The most important functions of the thesis project are to apply for an employee from among registered users, for example on the basis of drug licenses. The app allows to send free shift notifications to employees. The approved shift is saved to the user as well as the management side of the personal calendar. On the management side, it is possible to make additional requests for shifts that go to empty calendar shifts. Once the user accepts the shift, it becomes the reserved shift for the user.

Keywords
Azure, Nexmo, MVC, Web application

SISÄLTÖ

1 TERMIT JA LYHENTEET

POC-Malli	Proof of concept, tietyn menetelmän tai idean osoittaminen toteutuskelpoiseksi.
MVC	Model-View-Controller. Ohjelmistoarkkitehtuuri, jossa koodi jaetaan kolmeen osaan. Model käsittelee malliluokkia, View osio näkymät ja Controller suoritettavat metodit.
C#	Ohjelmointikieli Visual Studiossa.
AZURE	Microsoftin luoma pilvipalvelu.
SQL Local DB	Tulee sanoista Structured Query Language. Paikallisesti luotu tietokanta jota voidaan hallinnoida esim. SQL Server management studion kautta.
SMS	Short Message Service eli käytännönläheisemmin tekstiviesti.
NEXMO SMS API	Ent. Nexmo nyk. Vonage SMS viestien rajapintapalvelu.
RAZOR	Asp.Net Coressa esitelty menetelmä, jossa koodia ja näkymää voidaan jakaa samalle sivulle.
BOOTSTRAP	CSS kehys responsiivisien mobiilisivujen luomiseen.
NOTIFICATION HUB	Palvelu, joka mahdollistaa ilmoitusten lähettämisen mobiililaitteille.
FCM	Firebase Cloud Messaging ent. GCM eli Google Cloud Messaging. Mobiililaitteilla käytettyjen notifikaatioiden välityspalvelin.

2 JOHDANTO

Opinnäytetyössä tavoitteena oli luoda POC-malli sovelluksesta, jonka avulla työntekijän on mahdollista kontrolloida omia lisätyövuorojaan helpommin ja vastaavasti taas työnantajan on helpompi löytää varatyöntekijä oman yrityksensä sisältä. Mikäli on tarve saada työntekijä toisen tilalle nopeasti esimerkiksi sairastapauksen vuoksi, sovellus mahdollistaa nopeamman ja yksinkertaisemman keinon tekijän löytämiseksi. Mahdollisia soitteluja työntekijältä toiselle ei täten tarvita. Sovellus toimii Azure-pilvipalvelussa testaamisen helpottamiseksi.

Digitalisaation myötä mobiililaitteet ovat yleistyneet runsaasti ja niiden hyödyntäminen niin työelämässäkin on yleistä. Alkukartoituksen myötä kävi ilmi, ettei vastaavanlaisia sovelluksia juurikaan ole markkinoilla. Tai jos on, niiden toiminnot eivät vastaa työantajapuolen vaatimuksia, mikä on käyttäjäystävällisyyden kannalta huono. Toisin sanoen kehitysvaiheessa FE eli käyttäjäpuoli on tehty joko vaikeaselkoiseksi tai se ei vastaa asiakastyytyväisyyteen. FE tulee sanoista Front End ja tarkoittaa selaimen päässä suoritettavaa koodia. Sovelluksen käyttäjäkunta on tässä tapauksessa erittäin laajaskaalaisesti jakautunut nuoresta vanhukseen, kuten esimerkiksi teollisuudessa tai sote-alalla.

Työn tilaajana toimi yksityishenkilö nimellisesti, mutta kehitin sitä omalle yritykselleni, jonka perustin syksyllä 2019. Sain vapaat kädet niin suunnittelun, toteutuksen kuin testaamisenkin osalta. Tämä mahdollisti vapaan lähestymistavan sovelluksen suhteen.

3 KÄYTETYT TEKNIIKAT

Tässä osiossa käydään läpi opinnäytetyössä käytettyjä tekniikoita.

3.1 Microsoft Visual Studio

Microsoftin luoma kehitysympäristö, joka sisältää lukuisia ohjelmointikieliä mm. C#, C++, Javascript, Python. Sillä voidaan luoda nettisivuja, nettisovelluksia, tietokoneohjelmia tai ns. Backend palveluja Azure-pilvipalvelun kautta.

Visual Studion kehityskaari alkoi vuodesta 1997, jolloin versio Visual Studio 97 julkaistiin. Seuraavat versiot julkaistiin noin kahden vuoden sykleissä. Vuodesta 2003 lähtien vuosiluku viittasi ohjelmistoversion julkaisuvuoteen. (Microsoft 2019.)

3.2 MVC arkkitehtuuri

Lyhenne MVC muodostuu sanoista Model-view-controller. Arkkitehtuurimalli, joka jakaa sovelluksen kolmeen osaan. Tämä helpottaa tiedon käsittelemistä, koska jokainen osiot jakautuvat tietokannasta saataviin datamalleihin(Model), ohjelmistossa suoritettavaan koodiin(Controller) ja palautettavaan näkymään(View). (Microsoft 2020)

3.3 .NET Framework

Microsoftin kehittämä viitekehys ohjelmistotuotantoon. Ensimmäinen versio .NET Framework 1.0 julkaistiin jo vuonna 2002 ja versio 4.8 vuonna 2019. Toisinkuin .NET Core joka on tarkoitettu cross-platform kehitykseen Framework on pelkästään Windows pohjaisten sovelluksien kehitykseen. (Microsoft 2020.)

3.4 C#

Ohjelmointikieli, joka pohjautuu C++ kieleen ja Javaan. Ohjelmointikieli esiteltiin ensimmäistä kertaa vuonna 2000 .NET eli DOT Netin julkaisun yhteydessä. Vuonna 2003 C# standartisoitiin ISO luokituksella eli se mahdollisti kielen yleistymisen. ISO standartisoinnin ansiosta C# ohjelmointikielestä on kehittynyt yksi käytetyimmistä ohjelmointikielistä. (Microsoft 2015.)

3.5 SignalR

Asp.Net kirjasto, jota voidaan hyödyntää reaaliaikaisen toiminnallisuuden lisäämiseksi web sovellukseen. Mahdollistaa reaaliaikaisen tiedon siirtämisen esim. serveriltä client koneelle, jolloin datapyynnöt toimivat ilman pyyntökatkoksia. Voidaan hyödyntää esim. Chat ohjelmistoissa. (Fletcher 2014.)

3.6 Nexmo SMS API

Entiseltä nimeltään Nexmo eli nykyinen Vonage on välityspalvelin, joka mahdollistaa puheluiden, tekstiviestien käyttämisen ohjelmistossa hyödyntäen Vonagen rajapintaa. Yleisimpiä tekniikoita välittää esim. suoramarkkinointia koskevia tiedotteita asiakkaille.

Tekniikkaa käytetään myös kaksisuuntaisessa tunnistaumisessa eli 2FA:ssa. 2FA tulee sanoista Two-factor authentication, jota käytetään esimerkiksi tietokonepeleissä tilin tunnistamiseen. Puheluiden ja tekstiviestien hinnat määräytyvät maakohtaisesti hinnaston mukaan. Yhtiöllä on useita patentteja, jotka on suojattu U.S Patent merkinnöillä. Yhtiöllä on myös useita alan palkintoja esim. CRM Excellence palkinto tai Cloud computing palkinto vuodelta 2019. (Vonage 2020.)

4 SUUNNITTELU

Tässä kappaleessa käydään läpi sovelluksen suunnittelua ja siihen vaikuttaneita asioita.

4.1 Alkukartoitus

Idea lisätyövuorosovelluksesta sai alkunsa keväällä 2019 Tuotekehitysprojekti-opintojakson tiimoilta. Jo silloin kävi ilmi, että vastaavaa tuotetta ei ole markkinoilla tai niiden toimintaperiaatteet eivät vastanneet kuluttajatyytyväisyyteen puutteellisuuden vuoksi, koska esimerkiksi hyväksytyt vuorot eivät tallentuneet tietokantaan.

Työn suunnittelu alkoi syksyllä 2019, jolloin laadin työsuunnitelman aiheesta. Jo aikaisessa vaiheessa kävi ilmi, että tuote ei olisi lukittuna ns. yhteen kategoriaan vaan sitä voisi hyödyntää kaikilla aloilla.

Päätavoitteenani oli alussa kehittää sovellusta sosiaali- ja terveysalalle, koska tutkimuksien ja uutislähteiden mukaan Sote alalla sairauspoissaolot ovat työelämän korkeimpia ja se tiettävästi kuormittaa myös työnantajien työtaakkaa. Kuormittavuuden takia, johon vaikuttavat sekalaiset työvuorot ja hoitajamitoitus potilasta kohti ovat pääsyynä äkillisiin sairauspoissaoloihin. (Nieminen 2020.)

Henkilökohtaisen kokemuksen myötä myös muilla aloilla kuten teollisuudessa on enemmän sääntö kuin poikkeus, että mahdolliset henkilöstövajeet ovat jokapäiväinen ongelma. Tämä voi johtua säästösyistä tai ihan vain siitä, että työntekijä on ilmoittamatta pois työpaikaltaan.

Nämä kaikki edellä mainitut asia vahvistivat sitä ajatusta, että tämän kaltaiselle ohjelmistolle olisi kysyntää ja siitä voisi kehkeytä jopa mahdollinen myyntituote, koska asiakas segmentti on laaja ja ei ole sidoksissa pelkästään sosiaali- ja terveysalaan.

4.2 Muutosvastarinnan vaikutukset

Suunnittelussa täytyi ottaa huomioon myös muutosvastarinnan vaikutukset sovelluksen käytettävyyteen ja siihen, miten hyvin sovellus otettaisiin vastaan kaupallisessa käytössä, jos se sinne asti joskus päätyisi. Muutosvastarinta yrityksessä on tervettä ja se kielii halusta kehittää yritystä eteenpäin ja kertoo myös halusta jaksaa nähdä epäkohtia asioissa.

Useimmiten yrityksen sisällä koettu vastarinta voidaan nähdä myös erittäin negatiivisena asiana ja avoimen palautteen käsittelyä saatetaan pitää erittäin rasittavana, koska ihmisten ja heidän ajatusten tarkoitusperiä ei tiedetä varmaksi. Osa saattaa aiheuttaa vastarintaa pelkästä närkästyksestä työtä tai yhteisöä kohtaan ja osa taas aidosta halusta korjata epäkohta työpaikalla.

Sama vastarinta koskee myös työn ohessa käytettäviä sovelluksia. Aina kun yrityksen sisällä otetaan käyttöön uusia sovelluksia tai järjestelmiä se käy läpi muutosvastarinnan ja sen aiheuttaman hyljeksinnän.

TTT eli Työ, Terveys, Turvallisuus -lehden julkaiseman artikkelin mukaan muutosvastarinta pitäisi nähdä myös samalla innovation lähteenä ja samalla tekstissä mainitaan siitä, miten se voidaan kanavoida myös innovaation lähteeksi. Sama pätee myös mahdollisiin valituksiin ohjelmiston suhteen eli kanavoidaan se muutosvastarinta vastaamaan parannusehdotusta ohjelmiston suhteen, niin saadaan parhaiten asiakkaalle sopiva tuote aikaiseksi ja kaikki muutokset tulevat suoraan käyttäjiltä. Tämän avulla muutovastarinta voidaan kääntää innovaativiseksi kehitysideaksi. (Hasu 2015.)

Yhtenä sosiaali- ja terveysalan hyvänä esimerkkinä voidaan pitää Hälytyskeskuslaitoksen hankintaa nimeltään ERICA, jota on kritisoitu työntekijöiden puolelta siitä, että mahdolliset sairastapaukset ja hälytykset luokitellaan liian avokätisesti kiireelliseksi. Tämä taas johtaa esimerkiksi ambulanssihenkilöstön työmäärän nousuun ja sitä kautta kyseenalaistamaan koko hankinnan olemassaolo ja tarpeellisuus. Toisaalta artikkelin mukaan kysyttäessä itse Häke-päivystäjältä järjestelmässä nähdään myös paljon hyvää ja se on koettu tarpeelliseksi. Asialla on yleensä kaksi puolta ja esimerkiksi ohjelmistollakin on ne molemmat. Tämä ohjelmistokehityksessä on juurikin tärkeää että hiotaan ominaisuuksia niin kauan että molempien puolien kannattajat saadaan tyytyväisiksi. Ohjelmistoa on yleensä helpompi muokata kuin ihmisten asenteita. (Husu 2019.)

4.3 Käyttöliittymän suunnittelu

Käyttöliittymän suunnittelussa on pyritty mahdollisimman helppoon ja käytännölliseen ulkoasuun, joka mahdollistaa helpon omaksumisen ohjelmiston osalta. Opinnäytetyössä suunniteltiin MVC-mallin mukainen web-sovellus, joka on tarpeeksi kevyt ulkoasultaan, mutta samalla sisältää tarvittavan määrän ominaisuuksia, jotta sovellus on käyttökelpoinen. Käyttöliittymän suunnittelussa on otettu viitteitä myös muista nettisivuista, jotka koettiin toimiviksi ja toteutuskelpoisiksi malliltaan.
Heti alussa alussa oli selvää, että web- sovellus tullaan suunnittelemaan MVC-mallilla, koska se on yksi yleisimmistä toteutustekniikoista web- sovellusta luodessa. MVC-mallin eduiksi voidaan laskea helpohko näkymien kontrollointi ja näkymien jakaminen.

4.4 Käyttäjänäkymät

Alussa suunniteltiin käyttäjänäkymät, jotka olisivat yhteiset ja ne jaettaisiin eri osioihin partial näkymän avulla, jolloin käyttäjän toiminta määräisi mitä eri osa-alueita sovelluksesta näytettäisiin hallinnointi käyttäjälle ja työntekijälle. Sovellus on tarkoitettu kuitenkin käytettäväksi myös mobiililaitteilla, joten näkymän piti skaalautua luontevasti myös pienempäänkin kuvakokoon.

Skaalautuvuus säilyttäen kuvasuhteen hoituisi käyttämällä sovelluksessa Bootstrap kirjastoa, joka on pääasiassa FE eli Front End mobiilikehitys kirjasto.

4.4.1 Hallinnointinäkymä

Tämä osio kuvastaa hallinnointinäkymää, jonka hallinnointi kuuluu yrityksessä esim. osastonhoitajalle. Suunnitteluvaiheesssa tehtiin havainne prototyyppejä niin kuvallisesti kuin ohjelmallisesti. Tämän tarkoituksena oli havainnollistaa mahdolliset yhtymäkohdat eri näkymien välillä eli suomeksi sanottuna rakentaa näkymistä/välilehdistä toisiaan tukevia ohjelmiston osa-alueita. Yhdeksi tärkeimmistä näkymistä valikoitui Kalenteri-näkymä, joka oli visuaalisesti miellyttävä ja samalla toiminnallisuudeltaan tarpeeksi riittävä tapahtumien hallintaan.

Tämä ajattelutapa tuki "Less is more" ajattelutapaa eli enemmän toisiaan tukevia osa-alueita ilman että näkymä puuroutuu välilehtitulvaan.

4.4.2 Kuluttajanäkymä

Tämän osion tarkoituksena oli kehittää näkymä työntekijöille ja tehdä siitä mahdollisimman yksinkertainen ja käytettävyydeltään helppo. Suunnitteluvaiheessa haluttiin tehdä maksimissaan viisi sivuinen näkymä, koska kuluttajan oli tarkoitus operoida sovellusta myös mahdollisesti mobiililaitteella.

4.5 Käyttöliittymän toiminnallisuus

Toiminnalisuutta käyttäjille näytettävien näkymien takana alettiin miettimään asiakas edellä, jolloin pystyttiin samaistumaan samaan tunnetilaan kuin oikea asiakas eli mikä on tarpeellista toteuttaa ja mitkä ominaisuudet tukevat sovellusta parhaiten.

Suunnittelun keskeisinä asioina pidettiin työntekijän mahdollisuutta hallita omia lisätyövuoroja kalenterin kautta ja hallinnointipuolen mahdollisuutta hakea työtehtävään oikeaa henkilöä. Toisena oli saada ilmoitus lisävuoroista mobiililaitteelle jollakin tavalla, koska tämä näytteli yhtä tärkeintä aspektia ja samalla se kertoisi käyttäjälle myös siitä, että sovelluksessa tapahtui jotain sillä välin, kun käyttäjä oli poissa. Kolmannen osa-alueen mietinnän kohteena oli mahdollistaa yhteydenpito myös sovelluksen sisällä, jolloin mahdolliset epäselvyydet työntekijän ja hallinnoinnin välillä voisi hoitaa sujuvasti ilman perinteistä soittelua tai tekstiviestien lähettelyä.

4.6 Tietokanta

Tietokannan suunnittelussa huomioon otettuja ja tärkeimpiä kriteerejä olivat tietokannan pitäminen mahdollisimman tiiviinä eli normalisoituna. Tietokantaan suunniteltiin testivaiheessa neljä taulua ja määritettiin pää- ja vierasavaimille viittaukset toisiin tauluihin, jotta yhdistettyä tietoa oli helpompi hakea.

Tietokannassa haluttiin tehdä taulut käyttäjille, henkilökohtaisille tiedoille, kalenterin tapahtumille, toimialueen paikkakunnille ja kirjautusmistiedoille, jolloin näkymien ohjautumista olisi helpompi ohjailla riippuen siitä onko työntekijä vai hallinnointipuolen käyttäjä.

Tietokannan normalisointia ohjasi myös se perusajatus siitä, että tietokannan koko ei saisi ylittää tiettyä megabitti rajaa, jos se aiotaan julkaista Azuren omaan pilvipalveluun ja luoda sinne oma tietokanta. Ilmaisversion tietokannan koko rajoittui 32 megabittiin.

Jo suunnitteluvaiheessa testauksessa selvisi, että kohtuullisella käyttäjämäärällä ja tietokannan normalisoinnin ajattelumallilla tietokanta olisi mahdollisimman monipuolinen ja datan määrä pysyisi sallituissa rajoissa.

4.7 Notifikaatiot/Ilmoitukset

Viimeisimpänä suunnittelussa täytyi ottaa huomioon mahdolliset tavat toteuttaa ilmoitus käyttäjälle saatavilla olevasta työvuorosta. Suunnitteluvaiheessa mietittiin mahdollisuutta käyttää hyödyksi Microsoft Azure-pilvipalvelun Notification Hubia. Suunnitteluvaiheessa selvisi se tosiasia, että Azuren hub-palvelu toimii vain välittäjäpalvelimen roolissa ja toimiakseen olisi vaatinut toisen ”oikean” hubin kuten FCM tai entiseltä nimeltään GCM.

Nämä kaksi olivat yleisimpimä tekniikoita välittää käyttäjälle ilmoitus mobiililaitteelle, vaikka vastaavia hub-palveluita on internetti pullollaan.

4.8 Julkaisu Azureen

Opinnäytetyön laatimaani suunnitelmaan kuului myös ohjelmiston ja sen tietokannan julkaiseminen Microsoftin pilvipalveluun Azureen. Syy Azuren valintaan johtui täysin siitä syystä, että se oli opiskelijalisenssillä ilmainen ja mahdollistaa helpon pääsyn ohjelmistoon. Sen lisäksi testaaminen ns. oikealla pilvessä pyörivällä ohjelmalla on helpompaa, koska se kuvaa parhaiten käyttäjäkokemusta.
Valintaan vaikuttivat myös ne tosiasiat, että Azureen julkaiseminen ja käyttöönotto voidaan toteuttaa suoraan Visual Studion kautta parilla napin painalluksella.

5 TOTEUTUS

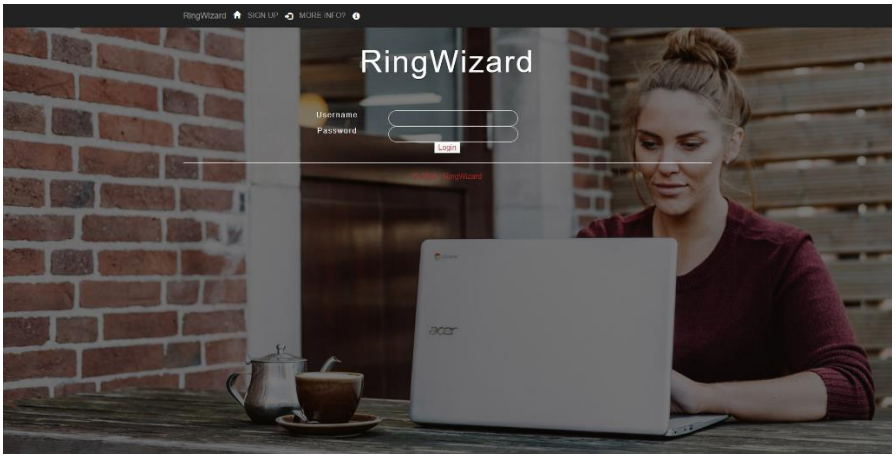
Tässä kappaleessa käydään läpi työn toteutuksessa käytyjä vaiheita ja ratkaisuja mihin opinnäytetyössä päädyttiin ja mitä muutoksia lopulliseen työhön piti tehdä. Osiossa käydään läpi myös vaihtoehtoisia kehitysmenetelmiä ja niitä syitä miksi päädyttiin tiettyihin menetelmiin ja mitä etuja muut menetelmät olisivat tarjonneet.

5.1 Käyttöliittymän toteutus

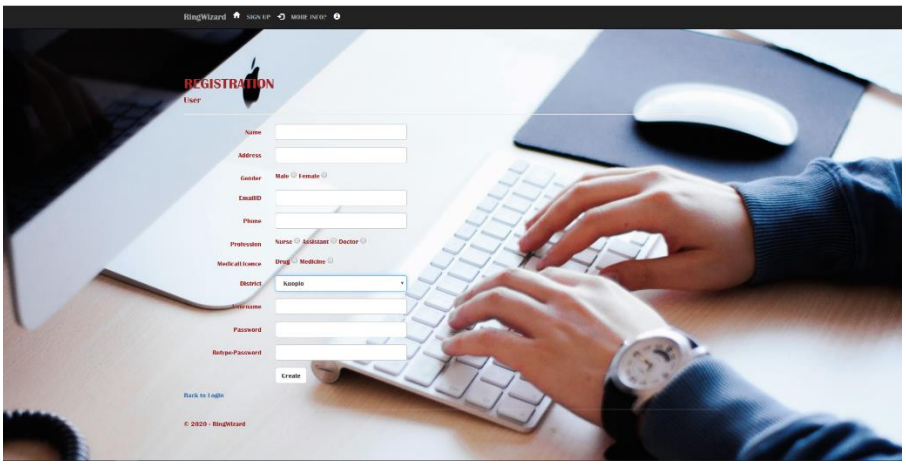
Käyttöliittymä toteutettiin .NET Frameworkia hyödyntäen Asp.Net MVC-mallilla, joka mahdollisti hyvän kehysrungon sovelluksen ulkoasua ja toiminnallisuutta ajatellen.

5.2 Käyttäjänäkymien toteutus

Käyttäjänäkymät päätettiin jakaa roolituksien mukaan Admin ja User näkymiin, jolloin rekisteröitynyt käyttäjä määritellään RoleID perusteella näkymäluokkaan. Admin tunnukset luodaan suoraan kantaan, jolloin mahdollisuus rekisteröityä Administratoriksi rekisteröintilomakkeen kautta on eliminoitu kokonaan pois. Näkymät jaotellaan kolmeen eri luokkaan. Yhteiset eli kirjautumissivu, rekisteröintisivu ja tutustumissivu, johon on toteutettu esipuhe siitä, mikä sovellus on ja mihin sitä käytetään. User-näkymät, jotka näkyvät pelkästään käyttäjille ja Admin-näkymä, joka näkyy vain Admin-tunnukset omaavalla henkilöllä. Sovelluksen CSS eli tyylitiedostoon on tehty muokkauksia fonttien, värin ja glyphigonien eli kuvakkeiden suhteen, jolloin käyttäjä näkee jokaisen välilehden yllä kuvakkeet, jotka lisäävät visuaalista näyttävyyttä ja parantavat navigointia sovelluksessa. Idea kuvakkeista tuli selailtuani eri nettisivuja ja useimmiten kuvakkeita esiintyi esimerkiksi nettikasinoiden sivuilla.



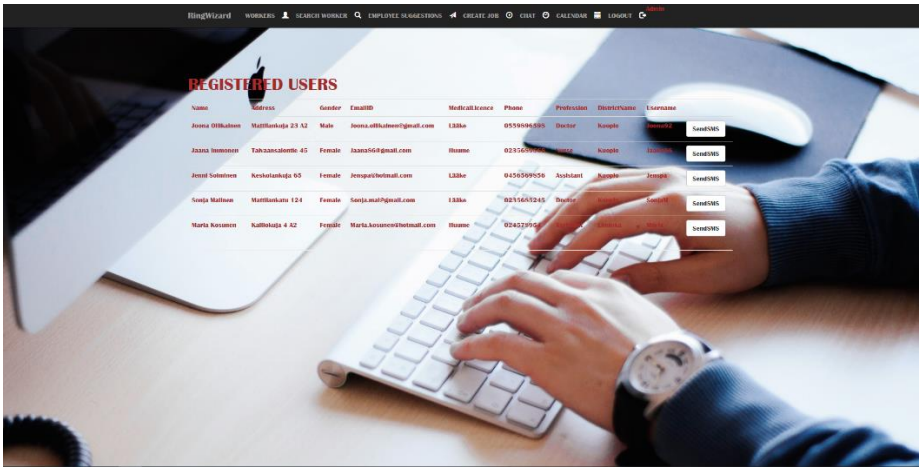
(Kuva 1. Kirjautumissivusto yhteisessä näkymässä.)



(Kuva 2. Rekisteröintilomake yhteisessä näkymässä, jossa tunnukset luodaan.)

5.2.1 Hallinnointinäkymä toteutus

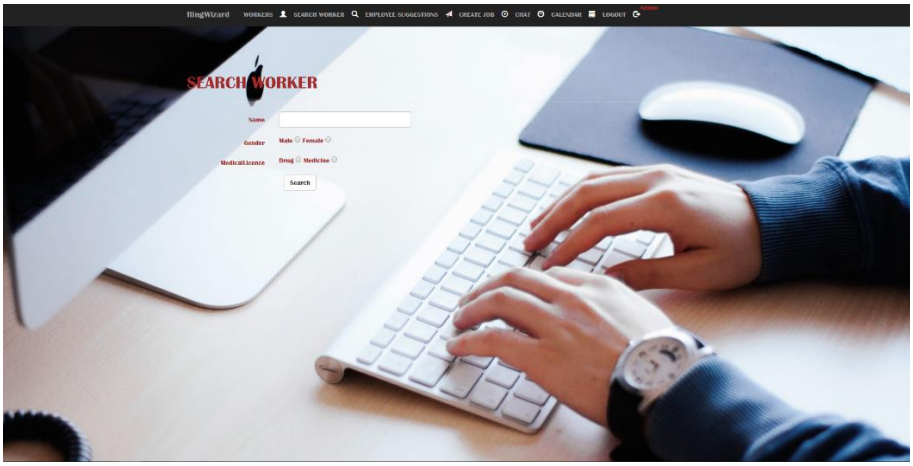
Admin käyttäjän näkymään päätettiin toteuttaa välilehdet Workers, Search Worker, Create Task, Chat ja Calendar. Kaikilla toiminnoilla on omat näkymät, jotka on määritetty LayoutAdmin kansioon samoin kuin kaikki Controllerit on määritetty AdminControllers nimisen kansion alle. Tämä selkeytti huomattavasti metodien ja näkymien hallinnointia. Workers näkymässä ladataan kaikki rekisteröityneet käyttäjät tietokannan User taulusta ja samalla luodaan heille dynaaminen SendSMS, jolloin tarvittaessa hallinnointi voi lähettää suoraan notifikaation/SMS viestin (Kuva 3). Tämä helpottaa tilannetta, jolloin tiedetään tietyn ihmisen tehneen tiettyä työtehtävää ja halutaan ilmoittaa mahdollisesta lisävuorosta suoraan hänelle. Numero generoituu suoraan Send-näkymään, jossa on kaksi tekstikenttää. Toinen numerolle ja toinen halutulle tekstille esimerkiksi ”Aamuvuoro Kys 06-14”.



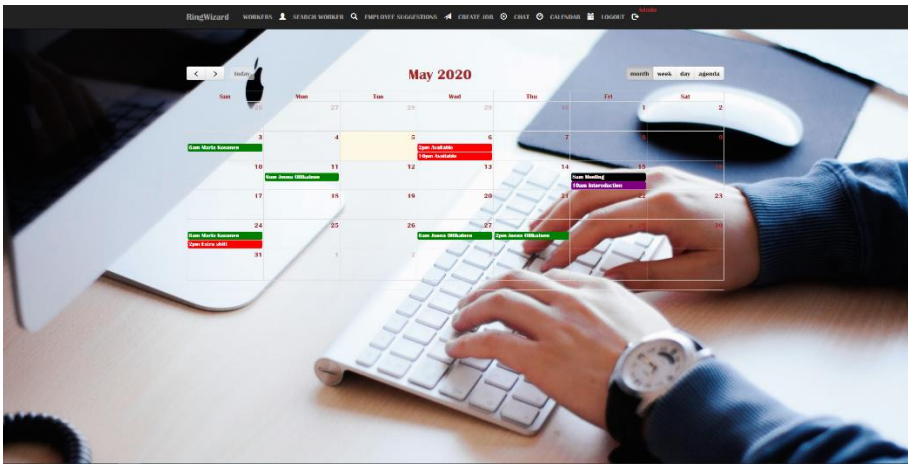
(Kuva 3. Rekisteröityneet käyttäjät Admin näkymässä ja mahdollisuus laittaa SMS suoraan.)

Sama ideologia pätee myös Search Worker -välilehdellä, jolloin sopivaa tekijää ei tiedetä ja halutaan hakea esimerkiksi huumeluvallista työntekijää (Kuva 4). Haun jälkeen päästään samaan lopputulokseen ja ilmoitusosioon. Kalenterielementti on toteutettu valmiilla komponentilla, jota varten luotu Calendar taulu, johon merkinnät tallentuvat eventteinä. Kalenteri mahdollistaa lisäämisen, poiston ja muokkauksen. Kalenteriin toteutettiin ominaisuus, joka mahdollistaa Adminin laittaminen avoimien työpyyntöjen tallentumisen käyttäjä näkymän Task -välilehdelle sekä Admin puolen omaan kalenteriin

värikoodilla punainen. Ideana tässä oli, että hallinnointipuoli voi laittaa vapaita vuoroja jakoon ja tätä kautta täydentää varsinaista työvuorosuunnittelua. Työntekijän hyväksyttyä vuoron eventiä muokataan lennosta, jolloin väri saa arvon vihreä ja subject arvon hyväksyjän nimi. Calendar tauluun tehtiin myös kentät IsAdmin ja IsPending, jolla määritettiin, onko kalenterimerkintä Adminin oma merkintä vai käyttäjälle menevä työpyyntö merkintä (Kuva 5).



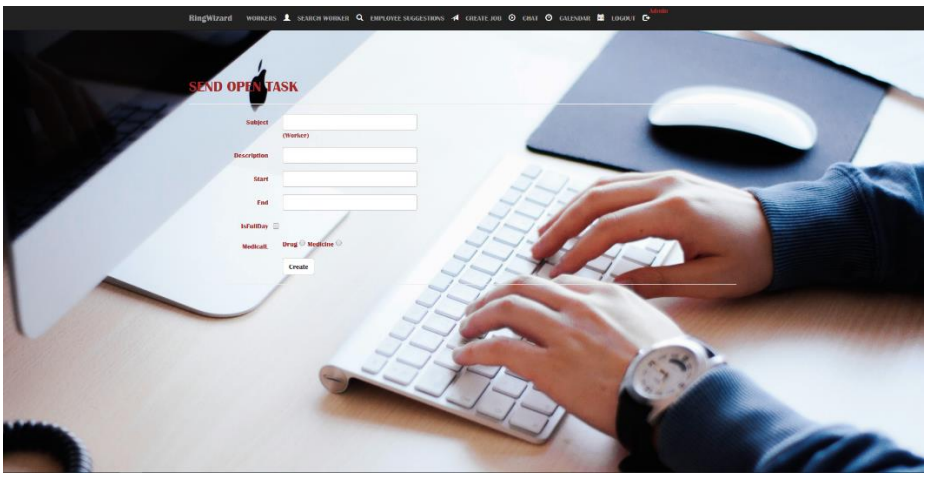
(Kuva 4. Haetaan työntekijä rekisteröityneiden käyttäjien joukosta.)



(Kuva 5. Admin kalenterinäkymä lisätyövuorojen hallinnointiin.)

Hallinnointi puolelle luotiin mahdollisuus muuttaa työvuoromerkintää niin, että varmistamattomat vuorot voidaan muuttaa työvuoropyynnöiksi kalenterin muokkaus näkymän kautta. Tämän tarkoituksena oli helpottaa tilannetta, jossa ei tiedetä tarvitaanko tietylle päivälle työntekijää vai ei.

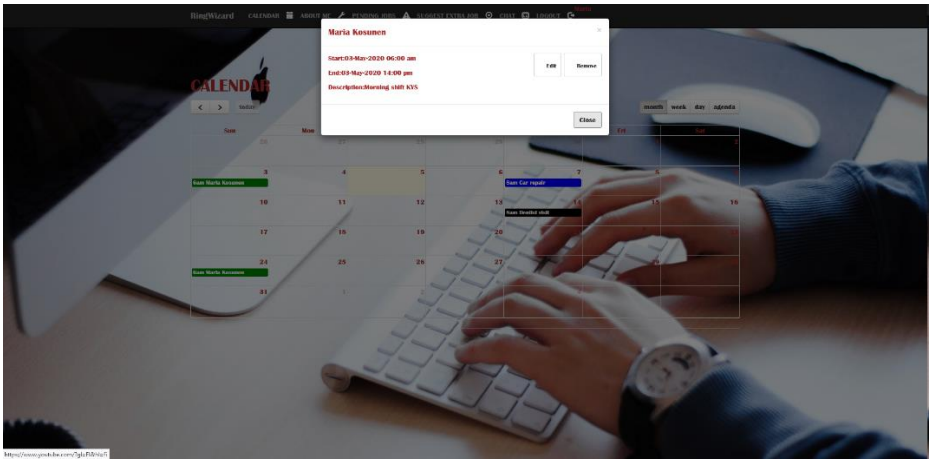
Avoimen työtehtävän luomista varten on hallinnointi puolelle tehty Create Job niminen välilehti. Tämä näkymän avulla kalenteriin saadaan lisättyä avoin tehtävä ja samalla lähetettyä se tehtävä käyttäjälle. (Kuva 6).



(Kuva 6. Avoimen työtehtävän luonti. Tehtävät siirtyvät käyttäjille ja Admin kalenteriin.)

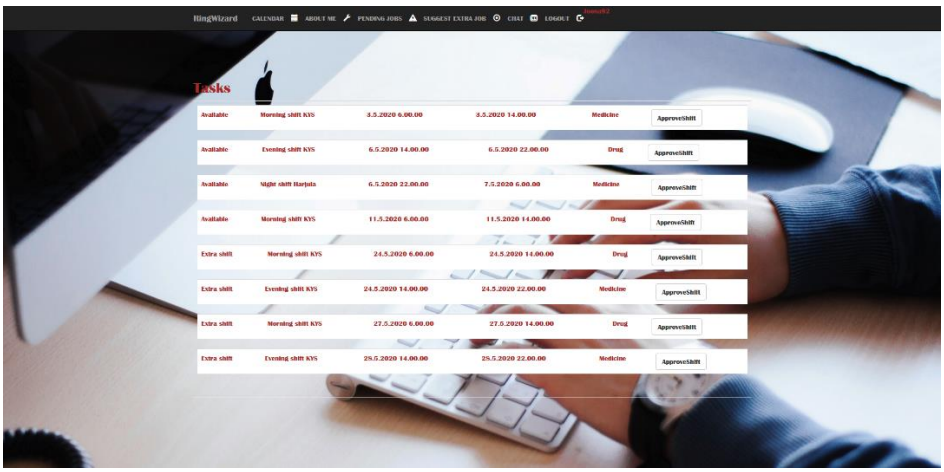
5.2.2 Kuluttajanäkymä toteutus

Käyttäjä puolen näkymään haluttiin tehdä osiot Calendar, About Me, Pending Jobs, Suggest Extra Job ja Chat. Kalenterin asetukset määräytyvät Calendar taulun samoilla Bool tyyppisillä muuttujilla, joilla määritetään hyväksyttyjen vuorojen näkyvyys aluksi vai Admin-näkymässä kunnes vuoron hyväksynnän jälkeen se siirretään osaksi myös työntekijän henkilökohtaiseen kalenteriin. Samalla haluttiin myös mahdollistaa työntekijän omat henkilökohtaiset merkinnät, jotta henkilön olisi helpompi sovittaa lisävuorot omiin tarpeisiin (Kuva 7).



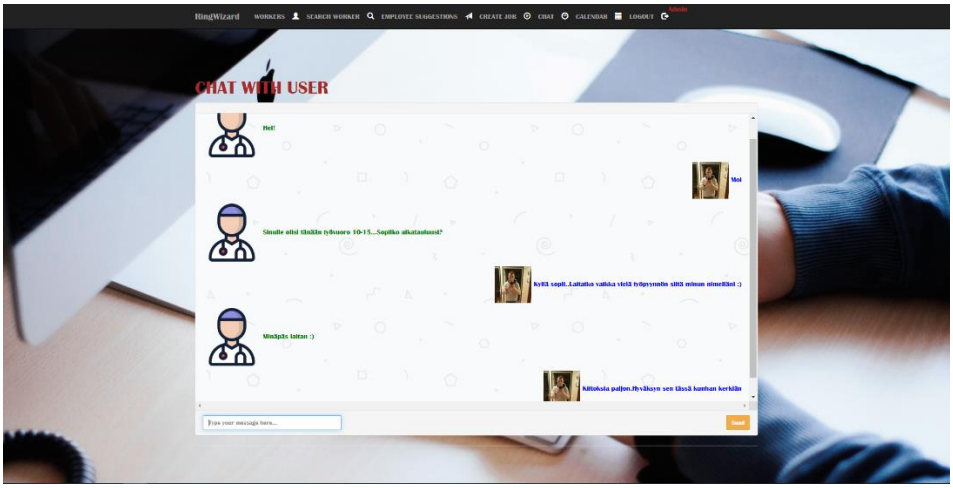
(Kuva 7. Työntekijän henkilökohtainen kalenteri, johon työtehtävät siirtyvät.)

SMS viestin saatuaan avoimet työtehtävät näkyvät käyttäjä puolen Tasks-välilehdellä, jossa ilmenee aihe, vuoro, kellonaika ja vaadittava lääkelupa. Task-välilehden avoimet työtehtävät ladataan listana näkymään ja lista päivittyy työtehtävien hyväksynnän mukaan. Toisinsanoen joka kerta kun joku hyväksyy vuoron, lista päivitetään ja poistetaan hyväksytty vuoro listalta (Kuva 8).



(Kuva 8. Lisätyövuorojen listaus ja mahdollisuus hyväksyä itselle sopivat vuorot.)

Chat ominaisuus toteutettiin hallinnointi puolelle sekä käyttäjä puolelle SignalR liitännäistä hyödyntäen. Chattia varten piti ladata liitännäinen Microsoft.AspNet.SignalR 2.4.1 ja määrittää samalla Startup.cs tiedostoon Microsoft Owin liitännäinen. Signal kansioon luotiin Signal Hub V2 niminen tiedosto, joka alustaa client yhteyden eri käyttäjien välillä ja mahdollistaa reaaliaikaisen chatin(Kuva 9). Tähän teknikkaan päädyttiin siitä syystä, että se on yksi käytetyimmästä Chat tekniikoista.



(Kuva 9. Mahdollisuus reaaliaikaiseen chattiin molemminpuolin.)

5.3 Tietokanta

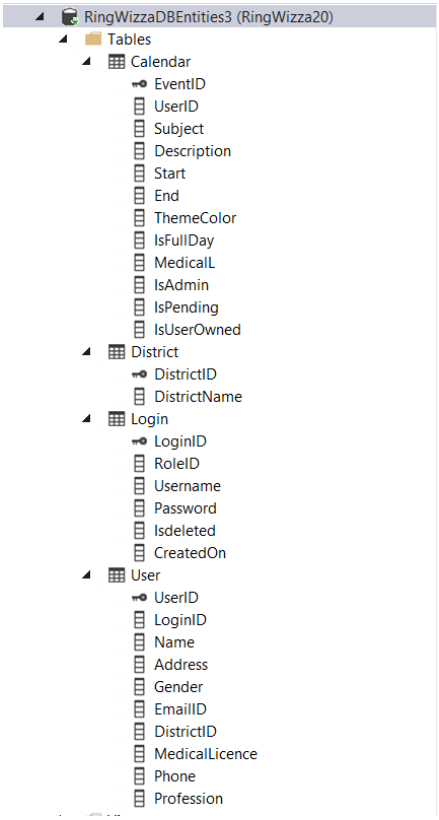
Tietokanta päätettiin toteuttaa paikallisesti testausta ja käyttöä varten ja todeta sen normalisointi riittäväksi ennen kuin se siirrettäisiin osaksi Azure-pilvipalvelun tietokantaa. Alussa suunnitellun ohjeistuksen mukaan tietokanta päätettiin pitää mahdollisimman yksinkertaisena sisältäen kuitenkin tarvittavat taulut ja sarakkeet, jotta tarvittava data saataisiiin talteen.

Tietokannan muokkaaminen ja luominen tapahtui SQL Management Studiolla, jonka totesin olevan mielekkäämpi tietokannan muokkaukseen kuin Visual Studion oman editorin. Tietokannan nimeksi määritettiin RingWizzaDB ja siihen luotiin neljä taulua.

Calendar taulu tarkoitettiin kalenterinkomponentin käyttöön ja se toimisi työvuorojen, henkilökohtaisten merkintöjen, oman työajanseurannan ja saapuvien työvuorojen hallintaan liittyvänä tauluna. District taulu taas päätettin hajauttaa User taulusta monesta moneen viittauksien vuoksi ja sen tarkoituksena pitää sisällään data ihmisten asuinpaikasta ja työyksiköstä. Login tauluun tallennettiin käyttäjien roolit, käyttäjänimet, salasanat ja kirjautumisID. Nämä tiedot tallennetaan Login tauluun siinä vaiheessa, kun käyttäjä rekisteröityy sovellukseen ensimmäisen kerran ja luo oman henkilökohtaisen tilin. User tauluun tallennetut tiedot sisältävät käyttäjän oleellisimmat tiedot käsittäen mm. nimen, osoitteen, sähköpostin, lääkelisenssi ja puhelinnumero.

Taulujen välille luotiin viittaukset pää- ja vierasavaimien avulla, jolloin data saatiin tallentumaan oikein taulujen välillä ja dublikaatteja ei päässyt syntymään. (Kuva 10.)

Yksi .Net Frameworkin ohjelmointia helpottavista puolista oli, että kannan luonnin jälkeen Modelit eli malliluokat sai luotua kätevästi ADO.NET Entity data model menetelmää hyödyntäen. Käytännössä tämä tarkoitti sitä, että luotiin taulu ja siitä malliluokka, jolloin data model generoi malliluokan suoraan taulusta ja siihen viittaavista tauluista. Malliluokkien tai tietokantataulun muuttaminen ei vaatinut jälkeenpäin muita toimenpiteitä kuin taulun tietojen muokkaamisen ja tauludiagramissa ”Update data model from database” komennon suorittamisen. Tällä menetelmällä mallliluokkien luominen oli nopeaa ja vaivantonta, mutta jälkeenpäin muokattujen malliluokkien datatyypit saattoivat silloin tällöin ns. jumiutua, jolloin näkymä eli View ei tunnistanut lisättyjä datatyyppejä. Tässä tilanteessa projektin rakentaminen eli buildaaminen korjasi ongelman.



(Kuva 10. Kantaan määritetyt taulut ja kolumnit sekä viittaukset.)

5.4 Ilmoitukset/Notifikaatiot

Suunnitteluvaiheessa mahdollisia potentiaalisia vaihtoehtoja oli käyttää Azuren Hub Notification palvelua, joka lähettäisi ilmoituksen, kun esim. tietokannan taulussa tapahtuu muutoksia. Tämä idea tai toisinsanoen toteutustapa päätettiin hylätä projektin edetessä syystä, että se olisi vaatinut FCM liitännäisen hubin ja toteutustavasta haluttiin suoraviivaisempi.

Toteutustapa päätettin suoraviivaistaa ja toteuttaa ilmoitukset perinteisellä SMS viestillä. Perinteisen tekstiviestin etuihin kuuluu se, että perinteisiin markkinointivälineisiin verrattuna tekstiviesti on tehokkain tapa tavoittaa kohderyhmä. Tutkimusten mukaan tekstiviesti on lähes ainoa markkinointiväline, joka luetaan lähes aina. Vaikka eletään vuotta 2020 kaikki eivät välttämättä omista älypuhelinta vaan turvautuvat perinteiseen kännykkään. Tämä oli otettava toteutuksen aikana huomioon, koska työpaikkojen ikärakenteet voivat vaihdella eri ikäryhmien välillä. (Tavoittaja 2019.)

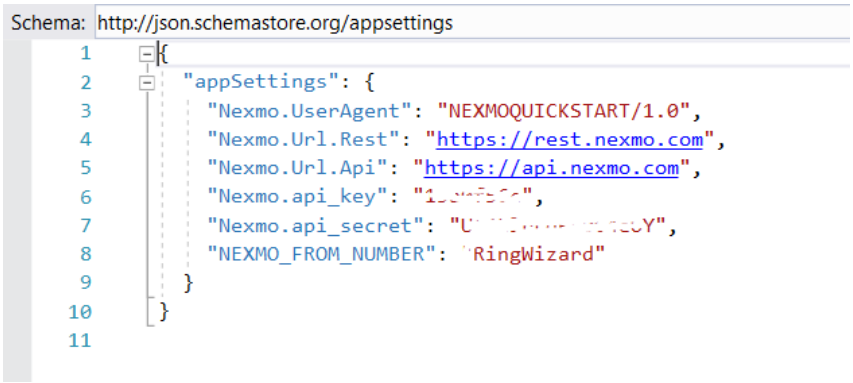
Tilastokeskuksen tuottaman tutkimuksen mukaan 25 - 34 vuotiaista 98 prosenttia käyttää internettiä matkapuhelimella, kun taas 55-64 vuotiaista vastaava luku on 84 prosenttia. Tämä jättää marginaalisen kohderyhmän auki mahdollisuudelle, että perinteiset Android/IOS ilmoitukset eivät tavoita kohderyhmää yhtä tehokkaasti kuin kaiken tavoittava perinteinen SMS. (SVT 2019.)

Nexmo SMS rajapinnan käyttöönotto sovelluksessa tapahtui luomalla tili palveluun, jolloin omalle henkilökohtaiselle tillille sai käyttöön kaksi euroa rajapinnan testaamiseen. Seuraavaksi luotiin AdminControllerin alle metodi Send() ja erillinen appsettings.json tiedosto johon tarvittavat rajapinnan tiedot tallennettiin. Näkymien eli Viewien alle luotiin oma näkymä Send, josta AdminControllerin Send() metodia kutsuttiin lähettäessä SMS viestin. Nexmo palvelun kautta hyödynnettyä rajapintaa varten piti ladata NuGet liitännäiset, jotka sai asennettua joko consolen kautta tai solution managerin kautta.

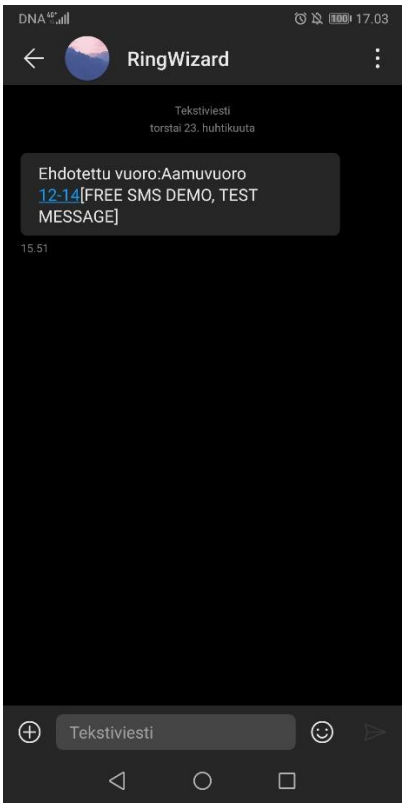
Liitännäisten vaatimuksena oli asentaa Nexmo.Csharp.Client versio 2.2.0 ja Microsoft.Extencions.Logging versio 1.0.1. Asennuksien jälkeen Solutionin kaikki liitännäiset piti päivittää yhteensopiviksi, koska tietty Framework versio ei yksinkertaisesti ollut yhteensopiva liitännäisen kanssa. Tämä aiheutti ongelmia koko projektin kanssa, koska aiemmin määritetyt liitännäiset oli määritetty tiettyyn ohjelmistoversioon ja päivityksen jälkeen tietyt etukäteen asennetut liitännäiset lakkasivat toimimasta tai toimivat puutteellisesti. Päänvaivaa aiheutti etenkin Bootstrap kirjaston päivitys versiosta 3.0 versioon 4.0. Bootstrapin päivityksen yhteydessä koko ohjelman Layout eli ulkoasu menetti muotonsa ja oli hajalla. Ongelmaan löytyi ratkaisu internetin ja version manuaalisesta päivittämisestä alaspäin. Tämä oli valitettavan laaja ongelma myös muiden kehittäjien keskuudessa, mikä ilmeni keskustelufoorumeja selatessa. Ilmeisesti version päivitys pyyhki kaikki esiasetetut asetukset pois ja asetti näkymiin uudet arvot ja asetukset.

Tilin luonnin yhdeydessä tilinhaltija sai henkilökohtaiset Nexmo.api.key ja Nexmo.api.secret avaimet jolla tili todennettiin henkilökohtaiseksi. Nämä avaimet toimivat määrittäjinä sille tilille kenen ”laskuun”

tekstiviestit lähetetään (Kuva 11). Tekstiviestien hinnoitteluperusteena toimi maakohtainen hinnoittelutaulukko ja operaattorit. Suomessa operaattorit veloittavat 0,07 euroa per tekstiviesti. Ilmaisversiossa pakollinen ”Demo” ilmoitus olisi poistunut sisäisellä ostolla pois (Kuva 12.)



(Kuva 11. Rajapinnan käyttöä vaativat yhteysavaimet)



(Kuva 12. Saapunut ilmoitus sovelluksesta määritettyyn numeroon)

Toimivaksi todettu Nexmo välityspalvelin valittiin kehityskäyttöön siitä syystä, että käyttöönotto oli helpompaa kuin esim. Twilio palvelulla, vaikka käytännässä periaate toiminnalle oli sama. Testikäyttöä helpotti myös se, että tosinkuin Twiliolla Nexmolla käyttöön sai ”testausrahaa” eikä vaadittu talletusta oikealla rahalla.

5.5 Julkaisu Azure pilvipalveluun

Suunnitelmien mukainen julkaisu Microsoftin ylläpitämään Azure-pilvipalveluun tapahtui suoraan koulun tiliä hyödyntäen ja julkaisun mahdollisti Visual Studion kautta helposti saatavilla ollut julkaisu vaihtoehto. Julkaisun voi tehdä kahdella tavalla eli joko Azure Portalin kautta luoda Azure App serviceen palvelu tai tehdä julkaisun Visual Studion kautta. Henkilökohtaisesti huomasin Visual Studion kautta tehdyn julkaisun huomattavasti helpommmaksi. Samalla luotiin myös tietokanta, johon myöhemmin tietokantaskripti myös ajettiin.

TechRadar sivuston listauksen mukaan Microsoftin Azure kuuluu parhaimmistoon viidestä listatusta pilvipalvelun tarjoajasta vaikkakin se soveltuu enemmänkin yrityskäyttöisten palvelinten pyörittämiseen kuin pelkän yksittäisen nettisivun. Yhtenä Microsoft Azuren etuina voidaan laskea yhtenäinen alustakieli eli toisinsanoen sillä ei ole väliä millä ohjelmointikielellä ohjelmisto tai palvelu on toteutettu, vaan kaikkia ohjelmistoja voidaan ajaa yhteinäiseltä alustalta. (Williams 2020.)

Muut listatut pilvipalveluntarjoajat ovat muun muassa Hostgator, Cloudways, Clook ja DigitalOcean. Kaikista listatuista vaihtoehdoista Hostgator soveltuu sivuoston mukaan parhaiten yksittäiseen kuten nettisivusovelluksen pyörittämiseen. Listauksessa Hostgator on sijoittunut toiseksi ja sen vahvuuksiin on lueteltu kilpailukykyiset pakettiratkaisut, sovelluksen peilaus eri servereille, jolloin viive laskee huomattavisti esim. käyttäjämäärän kasvaessa. Sovelluksen vaativuuden kasvaessa kasvaessa myös virtuaalikoneen prosessorien määrää voidaan kasvattaa ns. lennosta, jolloin sivusto pyörii paremmin.

Microsoft Azuren valinta oli itselleni helppo, sillä se oli ainoa pilvipohjainen ratkaisu, joka oli tullut koulussa tutuksi. Julkaisu itsessään ei tuottanut ongelmia vaan se oli kaiken kaikkiaan erittäin yksinkertainen toimenpide, joka sisälsi vain muutaman konffauksen. Tietokannan muokkaaminen tapahtui ottamalla SQL Management Studiolla paikallisesta tietokannasta Sql skripti talteen ja kirjautumalla Azuressa tietokannan Preview editoriin tunnuksilla, jotka luodaan App servicen yhteydessä. Tietokannan luominen manuaalisesti skriptin avulla oli pakollista, koska en itse henkilökohtaisesti löytänyt ohjetta sille, miten olemassa olevan paikallisen tietokannan saisi siirrettyä Azureen kokonaisuudessaan muuten kuin ns. kloonaamalla.

6 TESTAUS

Ohjelmiston testaus kulki käsi kädessä ohjelmiston kehityksen kanssa. Heti alkumetreiltä oli tärkeää testata ohjelmiston toimivuus Azure-pilvipalvelussa, sillä haluttiin poissulkea mahdollisuus sille, että ohjelmiston valmistuttua se ei käytännössä olisi ollut toimiva pilvipalvelussa. Azureen julkaisu oli yksi tärkeimmistä projektin vaiheista, koska silloin se siirtyi ns. kellariprojektista testausvaiheeseen ja ”oikeaksi” sovellukseksi.

Yksi kohdatuista testauksen ongelmista liittyi sovelluksessa olevaan kalenterikomponenttiin ja sen toimintaan Azuressa. Keskustelufoorumeja selatessa kävi ilmi, että Azuressa toimivia kalentereja oli vain muutama ja niiden testaaminen sekä paikallisesti toimivana että Azuressa oli oikeastaan ainoita toimivuuden kannalta tärkeitä elementtejä. Muut osa-alueet oli testattu ja tiedettiin toimiviksi niin paikallisesti kuin pilvessäkin.

Kalenterikomponentin ongelma liittyi Ajax kutsuun ensimmäisen kerran suorittaessa, jolloin se ei latatunut kunnolla ja saattoi jättää osan komponenteista lataamatta. Sovellusta ohjelmoitaessa käytettiin Visual Studion omaa Debugger ominaisuutta, jolla saatiin testattua eri näkymien väliltä metodit. Tämä helpotti tekemistä, koska nähtiin suoraan esimerkiksi Create Job napin kenttien arvot. Toisinsanoen saatiin selville suoraan ilman ongelmia, että yritetäänkö kantaan viedä tyhjää tietoa vai saivatko kentät oikeasti niille sopivat arvot.

7 JATKOKEHITYS

Opinnäytetyön ollessa lähes valmis, oli pakko ajatella mahdollisia kehityskohteita ja mahdollisia uusia ominaisuuksia. Sovellukselle mahdollisia lisäominaisuuksia voisivat olla esimerkiksi perinteisen tekstiviestin lisäksi lähetettävä notifikaatio, jolloin saavutettaisiin maksimaalinen tavoitettavuus asiakkaan ja käyttäjän suhteen. Samalla mahdolliset sivuston sisäiset ilmoitukset esimerkiksi saapuneista chat viesteistä tai sivuston sisäisestä liikenteestä voitaisiin toteuttaa helposti ja se toisi lisäarvoa sovellukselle paremman käyttäjäkokemuksen ansiosta.

Lisäksi sovellukseen voisi yhdistää työajanseurannan tai muun vastaavan hyötyä tuovan lisäominaisuuden, joka ei kuitenkaan olisi liian monimutkainen käyttäjälle tai liian paljon sovellusta kuormittava. Sovellukseen voitaisiin lisätä työaikojen vertailu, joka vähentäisi päällekkäisien vuorojen tarjoamista työntekijöille. Käytännössä tämä tarkoittaisi, että tietokannassa tehtäisiin vertailu eri aikojen välillä, jolloin saataisiin selville, onko työntekijällä jo töitä vai onko hän mahdollisesti vapaa tekemään lisävuoroja. Hallinnointipuolen käyttäjällä olisi helpompi työ ehdottaa eri vuoroja henkilöille, jos näkisi onko tämä vapaalla vai ei. Esimerkisi työntekijälistauksessa värikoodi ilmaisisi mahdolliset muutokset. Perinteisellä liikennevalo taktiikalla eli vihreä, keltainen, punainen olisivat ne ilmaisimet ja värit, jotka ovat jokaiselle tuttuja ilman turhia selittelyjä ja tarkoitusperiä. Punainen tarkoittaisi vuorossa olevaa tekijää, keltainen poissaolevaa ja vihreä taas vapaalla olevaa tekijää.

Mahdollisesti sovelluksen ulkoasua voitaisiin muokata tulevaisuudessa vastaamaan enemmän nykyaikaista ulkoasua sekä asiakasyrityksen vaatimuksia ja tarpeita, jolloin se olisi enemmän asiakaskohtaisesti räätälöity sovellus. Ulkoasun muokkaus ei vaatisi kuin CSS tyylitiedoston muokkausta. Esimerkkinä tällä hetkellä sovelluksen ulkoasu vastaa yleissiistiä ulkoasua, mutta ei ole sidoksissa mihinkään tiettyyn työntekijäryhmään. Tulevaisuudessa sovelluksen menestymisen edellytyksenä on tehdä se vastaamaan nykyaikaisia standardeja. Sovelluksen ulkoasu muuttuisi dynaamisesti sen mukaan mihin työntekijäryhmään rekisteröitynyt käyttäjä kuuluu esimerkiksi ravitsemis- ja matkailualan käyttäjä näkisi omilla sivuillaan pelkästään omaan alaan liittyvää sisältöä tarkoittaen tällä esimerkiksi taustakuvaa ja sivuston sisältöä ja mahdollsesti myös eri näkymien avautumista. Tämä voitaisiin tehdä kantamuutoksilla ja esimerkiksi Bool tyyppisillä muuttujilla ja Partial View elementeillä.

Osittainen AD eli aktiivihakemisto migraation luominen sovellukseen mahdollistaisi myös olemassa olevien työntekijöiden listaamisen käyttäjäryhmään ilman ylimääräisiä rekisteröitymisiä. Siinä tapauksessa nykyinen sovellus, joka toimii tällä hetkellä enemmänkin ulkopaikkakunnalta muuttaneiden ja lisätyöntekijöiden työsovelluksena voitaisiin muuttaa yrityksen sisäiseen käyttöön sopivaksi ja se toimisi osana AD järjestelmää. Kaikki tarvittavat tiedot saataisiin helposti ryhmittäin ja nimikkeittäin yrityksen omasta tietokannasta sovelluksen tietokantaan. Tämä tullaan tekemään mahdollisesti tulevaisuudessa.

Alkuperäisessä epävirallisessa suunnitelmassa oli tehdä ohjelmaan näkymä, joka näyttäisi alakohtaiset työpaikkailmoitukset ja tukisi näin eri käyttäjien työllistymismahdollisuuksia ja lisäisi käyttäjien tietoi-

suutta oman alan työpaikoista. Tämä edesauttaisi työllistymistä omalla alalla, mutta se päätettiin jättää pois nykyisestä suunnitelmasta pois. Syystä että sen koettiin olevan epäolennainen osa nykyisessä sovelluksessa. Tämä näkymä malli voisi sopia paremmin jatkokehitettyyn sovellukseen, joka toimisi dynaamisesti käyttäryhmien välillä. Edellisissä kappaleissa viitattujen eri käyttäryhmien esimerkiksi MaRa-alan tapauksessa.

Aikaisemmin keväällä Tuotekehitysprojektin aikoihin mietittyä ideaa voisi soveltaa osana tätä sovellusta. Kyseessä oli Tinder tyyppinen sovellusidea, jossa rakkauden sijaan yhdistettäisiin työnhakija ja työnantaja. Tämä mahdollistaisi reaaliaikaisen työhaastattelun esimerkiksi FaceTimen avulla. Työnhakija selaisi työpaikkailmoituksia ja jakaisi tykkäyksiä. Samoin tekisi myös työnantajapuoli ja kun tykkäykset osuisivat molemminpuolisesti niin molemmilla osapuolilla olisi mahdollisuus saada infoa toisistaan mahdollsimman nopeasti ilman nykyistä byrokratian rattaiden hidastelua, joka pitää sisällään sähköpostin lähettelemisiä. Tämän sovelluksen ideana oli poistaa se mahdollisuus, että kyvykkään hakijan työura saattoi tyssätä pakolliseen sähköpostiin, joka ei koskaan tavoittanut oikeaa henkilöä.

Kaikki edellä mainitut kehitysideat ovat toteutuskelpoisia, mutta suurin ongelma on roolituksen miettimisessä eli mikä on se kategoria tai ryhmä, kenet sovelluksella aikoo tavoittaa. Kuten projektin aikana oli tapahtua eli sovellus meinasi paisua liian suureksi unohtaen todellisen käyttötarkoituksensa. Pienillä muutoksilla ja oikeilla ratkaisuilla sovellukseen lisättävät toiminnot voisivat nostaa sen lisäarvoa markkinoilla ja mahdollistaa samalla sen potentiaalisen tehokkuuden ja edelleen tukea aiemmin todettua ”Less is more” ajattelutapaa.

On helppoa ajatella, että paljon ominaisuuksia tuo suosiota tai lisää käyttäjäkuntaa, mutta pienetkin asiat ja muutokset oikein tehtynä voivat olla se avainasia, joka nostavat sovelluksen muiden yläpuolelle.

8 POHDINTA

Mielestäni opinnäytetyö kokonaisuutena onnistui hyvin, vaikkakin mutkia oli matkassa. Opinnäytetyötä tehdessä tuli tutuksi projektin aikataulutukset niin hyvässä kuin pahassakin. Projektin alkuvaiheilla olisi pitänyt tarttua tiukemmin aiheeseen ja siihen mitä mahdollisisia ongelmakohtia sovelluksen kehittämisessä ja testaamisessa saattaa olla edessä.

Projektin ensiaskeleissa olisi pitänyt tarttua tiukemmin suunnitelmalliseen kehitykseen ja keskittyä suoraan ominaisuuksiin, jotka tukivat sovellusta. Toisinsanoen alkuvaiheessa mietittiin liikaa ominaisuuksia niin määrältään kuin laadultaan. Tämä aiheutti projektin alkuvaiheessa hapuilua, joka kulutti liikaa aikaa ja tuotti sitä kautta päänvaivaa.

Projektin keskivaiheilla kehityssuunta alkoi hahmottumaan paremmin ja ideat pystyttin paremmin keskittämään kasaan. Sovelluksen kokonaiskuva ja siihen lisättävien ominaisuuksien tärkeys alkoi hahmottumaan paremmin. Toistojen kautta MVC malli tuli tutuksi ja alun tutustumisen jälkeen muokkaaminen ei tuottanut sen suurempia ongelmia. Kaikki halutut komponentit saatiin toimimaan halutulla tavalla ja lopputulokseen olen tyytyväinen.

Loppua kohden projekti alkoi hahmottumaan paremmin, jolloin toisiaan tukevia ominaisuuksia pystyi ajattelemaan selkeämmin ja miettimään niitä realiteettejä käyttäjän kannalta. Seuraavanlaisiin kysymyksiin pohdin vastausta mielessäni: ”Käyttäisinkö minä tätä sovellusta”, ”Tarvitaanko jotain lisää”, ”Ketä tämä sovellus palvelee tai tulee palvelemaan”. Kaikkia näitä asioita miettiessäni päädyin seuraavaan. Niin kauan, kun tekniikka kehittyy ei sovellus tule olemaan täysin valmis ja toiseksi täysin valmis sovellus ei ole koskaan hyväksi. Tämä johtuu ajatuksesta, että niin kauan, kun tarmoa kehittää sovellusta, on sovellus vielä kehitysvaiheessa eikä täysin valmis. Kaiken kaikkiaan olen tyytyväinen lopputulokseen ja tästä on hyvä lähteä jatkojalostamaan tuotetta.

LÄHTEET JA TUOTETUT AINEISTOT

FLETCHER, P. (2014). Introduction to SignalR. [Verkkojulkaisu]. Haettu 15.5.2020 osoitteesta Microsoft.com
https://docs.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr

HASU, M (2015). Muutosvastarinta voi kääntyä innovaatioiksi. [Blogiteksti]. Haettu 20.5.2020 osoitteesta tttlehti.fi
https://www.tttlehti.fi/muutosvastarinta-voi-kaantya-innovaatioiksi/

HUSU, P (2019). Hätäkeskukset. [Uutinen]. Haettu 18.5.2020 osoitteesta Yle.fi
https://yle.fi/uutiset/3-10835053

MICROSOFT (2019). *Welcome to the Visual Studio IDE*. [Verkkojulkaisu]. Haettu 10.5.2020 osoitteesta Microsoft.com: https://docs.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2019

MICROSOFT (2020). ASP.NET MVC Pattern. [Verkkojulkaisu] Haettu 17.5.2020 osoitteesta Microsoft.com:
https://dotnet.microsoft.com/apps/aspnet/mvc

MICROSOFT (2020). What is .NET Framework. [Verkkojulkaisu]. Haettu 15.5.2020 osoitteesta Microsoft.com
https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet-framework

MICROSOFT (2015). Introduction to the C# language and the .NET framework. [Verkkojulkaisu]. Haettu 15.5.2020 osoitteesta Microsoft.com
https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework

NIEMINEN, S (2020). Sairauspoissaolo. [Uutinen] Haettu 16.5.2020 osoitteesta Yle.fi
https://yle.fi/uutiset/3-11190167

SUOMEN VIRALLINEN TILASTO (SVT): Väestön tieto- ja viestintätekniikan käyttö [Verkkojulkaisu]. ISSN=2341-8699. 2019, 1. Suomalaisten internetin käyttö 2019. Helsinki: Tilastokeskus [viitattu: 24.5.2020]. Saantitapa: http://www.stat.fi/til/sutivi/2019/sutivi_2019_2019-11-07_kat_001_fi.html

TAVOITTAJA (2019). Tekstiviestimarkkinointi. [Verkkosivu]. Haettu 21.5.2020 osoitteesta Tavoittaja.fi
https://www.tavoittaja.fi/tekstiviestimarkkinointi

VONAGE (2020). About the new Vonage. [Verkkojulkaisu]. Haettu 16.5.2020 osoitteesta Vonage.com
https://www.vonage.com/about-us/?icmp=mainnav_aboutus_overview

WILLIAMS, M (2020). Best cloud hosting services in 2020. [Verkkojulkaisu] Haettu 20.5.2020 osoitteesta Techradar.com
https://www.techradar.com/news/best-cloud-hosting-providers